في الإصدارات السابقة من Net Framework. كان التعامل مع الـ XML (**Extensible Markup Language**) عملية بطيئة و صعبة و كان التعامل معها يتبع لما يسمى W3C DOM API (سنتكلم عنها لاحقاً) حيث كان على المبرمج في هذا الـ model أن يقوم بإنشاء مستند XML دائماً حتى و إن لم يكن لديه سوى عنصر XML وحيد و هذا يعرف بـ document-centric و هذا من الأمور المزعجة فعلاً.

بعد أن صدرت Net Framework 3.5. اختفت معظم المشاكل السابقة في التعامل مع لغة XML و أصبح من الممكن كتابة عناصر دون الحاجة إلى إنشاء مستند كامل أي element-centric بالإضافة إلى الاستفادة من الأنواع المجهولة للمتحولات (anonymous types) و إمكانية استخدام تقنية LINQ مع عناصر ومستندات XML أي أصبح إنشاء ملفات هذه اللغة و التعامل أسهل و أسرع من أي وقت مضى .

حسناً لنبدأ الآن ببعض الأمثلة ، لدينا ملف XML التالي و سنحاول كتابته أولاً باستخدام DOM API و من ثم باستخدام الصفوف التي توفرها لنا LINQ لنرى الفارق بينهما:

```
<BookParticipants>
  <BookParticipant type="Author">
    <FirstName>Joe</FirstName>
    <LastName>Rattz</LastName>
  </BookParticipant>
  <BookParticipant type="Editor">
    <FirstName>Ewan</FirstName>
    <LastName>Buckingham</LastName>
  </BookParticipant>
</BookParticipants>
```

الملف باستخدام DOM API :

```
            XmlElement xmlBookParticipant;
            XmlAttribute xmlParticipantType;
            XmlElement xmlFirstName;
            XmlElement xmlLastName;
            //  First, I must build an XML document.
            XmlDocument xmlDoc = new XmlDocument();
            //  I'll create the root element and add it to the document.
            XmlElement xmlBookParticipants =
xmlDoc.CreateElement("BookParticipants");
            xmlDoc.AppendChild(xmlBookParticipants);
            //  I'll create a participant and add it to the book participants
list.
            xmlBookParticipant = xmlDoc.CreateElement("BookParticipant");
            xmlParticipantType = xmlDoc.CreateAttribute("type");
```

```
            xmlParticipantType.InnerText = "Author";
            xmlBookParticipant.Attributes.Append(xmlParticipantType);
            xmlFirstName = xmlDoc.CreateElement("FirstName");
            xmlFirstName.InnerText = "Joe";
            xmlBookParticipant.AppendChild(xmlFirstName);
            xmlLastName = xmlDoc.CreateElement("LastName");
            xmlLastName.InnerText = "Rattz";
            xmlBookParticipant.AppendChild(xmlLastName);
            xmlBookParticipants.AppendChild(xmlBookParticipant);
            //  I'll create another participant and add it to the book
participants list.
            xmlBookParticipant = xmlDoc.CreateElement("BookParticipant");
            xmlParticipantType = xmlDoc.CreateAttribute("type");
            xmlParticipantType.InnerText = "Editor";
            xmlBookParticipant.Attributes.Append(xmlParticipantType);
            xmlFirstName = xmlDoc.CreateElement("FirstName");
            xmlFirstName.InnerText = "Ewan";
            xmlBookParticipant.AppendChild(xmlFirstName);
            xmlLastName = xmlDoc.CreateElement("LastName");
            xmlLastName.InnerText = "Buckingham";
            xmlBookParticipant.AppendChild(xmlLastName);
            xmlBookParticipants.AppendChild(xmlBookParticipant);
```

الملف باستخدام LINQ :

```
            XElement xBookParticipants =
                  new XElement("BookParticipants",
                    new XElement("BookParticipant",
                      new XAttribute("type", "Author"),
                      new XElement("FirstName", "Joe"),
                      new XElement("LastName", "Rattz")),
                    new XElement("BookParticipant",
                      new XAttribute("type", "Editor"),
                      new XElement("FirstName", "Ewan"),
                      new XElement("LastName", "Buckingham")));
            Console.WriteLine(xBookParticipants.ToString());
```

أعتقد أن الصورة لديكم الآن أصبحت واضحة أكثر .. لاحظوا صعوبة الكتابة بالطريقة الأولى بالإضافة إلى أنه من الصعب جداً توقع عناصر XML التي ستنتج بينما في الطريقة الثانية سنكتب أسطراً أقل بالإضافة إلى أنه أصبح من الواضح جداً شكل الخرج الناتج.

كما سبق و قلت في المقدمة فإنك إذا أردت إنشاء عنصر XML وحيد باستخدام W3C DOM model API فإنك ستكون مضطراً إلى إنشاء مستند XML جديد كما يلي :

```
// With the original W3C DOM API, you could not simply create an XML element,
XmlElement; you must
// have an XML document, XmlDocument, from which to create it:

XmlDocument myXmlDoc = new XmlDocument();
XmlElement xmlBookParticipant1 = myXmlDoc.CreateElement("BookParticipant");
```

أما الآن مع LINQ فأنت غير مضطر لذلك و تستطيع إنشاء عنصر وحيد بسهولة :

```
// The new LINQ-enabled XML API allows you to instantiate an element itself
without creating an XML document:

XElement xeBookParticipant = new XElement("BookParticipant");
```

حسنا و ماذا إذا كنا نريد مستند XML و ليس عنصراً فقط ؟ المثال التالي سيجيب عن هذا التساؤل:

```
XDocument doc = new XDocument(
   new XDeclaration("1.0", "utf-8", "yes"),
   new XComment("Sample RSS Feed"),
   new XElement("rss",
            new XAttribute("version", "2.0"),
            new XElement("channel",
            new XElement("title", "RSS Channel Title"),
            new XElement("description", "RSS Channel Description."),
            new XElement("link", "http://aspiring-technology.com"),
            new XElement("item",
                   new XElement("title", "First article title"),
                   new XElement("description", "First Article Description"),
                   new XElement("pubDate", DateTime.Now.ToUniversalTime()),
                   new XElement("guid", Guid.NewGuid())),
            new XElement("item",
                   new XElement("title", "Second article title"),
                   new XElement("description", "Second Article Description"),
                   new XElement("pubDate", DateTime.Now.ToUniversalTime()),
                            new XElement("guid", Guid.NewGuid()))
                        )
                    )
                 );
```

حيث:

XDocument لإنشاء مستند XML جديد.

XElement لإنشاء عنصر XML جديد (قد يكون داخل عنصر آخر).

XComment لإضافة التعليقات.

XAttribute لإضافة صفة لعنصر ما.

قمنا بكتابة ملف XML و لكن لن نستفيد شيئاً ما لم نكن قادرين على تخزينه و تحميله لاحقاً لذلك لتخزين الملف السابق نستخدم التابعSave كما يلي :

```
doc.Save(@"E:\file.xml");
```

كل ما نحتاجه هو تمرير المسار الذي نريد التخزين فيه إلى التابعSave . لاحظ أيضاً أن هذا التابع هو instance method أي نستدعيه من أجل Object معين.

عندما نفتح الملف الذي حفظناه لتونا سنشاهد أنه قد تمت كتابته بتنسيقXML الذي اعتدنا عليه و لكن يمكننا أن نلغي هذه الخاصية و نحفظ الملف كتسلسل نصي عادي و ذلك باستخدام SaveOptions و اختيار DisableFormatting كما يلي :

```
doc.Save(@"E:\file.xml",SaveOptions.DisableFormatting);
```

و سيكون الملف كما توقعتم أي :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><!--Sample RSS Feed--
><rss version="2.0"><channel><title>RSS Channel Title</title><description>RSS
Channel Description.</description><link>http://aspiring-
technology.com</link><item><title>First article
title</title><description>First Article
Description</description><pubDate>2010-03-
19T21:06:11.307319Z</pubDate><guid>ce13e62b-fec4-4c89-88cc-
bdf52127a4bf</guid></item><item><title>Second article
title</title><description>Second Article
Description</description><pubDate>2010-03-
19T21:06:11.3093192Z</pubDate><guid>606ddb72-7806-4ed1-910d-
95cd0eb5a0d9</guid></item></channel></rss>
```

لا أعتقد أن أحداً يفضل شكل الملف هكذا ! حسنا لنعد قليلاً إلى المثال الأول عندما تكلمنا عن DOM API و لنحاول طباعة الملف على الشاشة لنشاهد كيف سيكون الخرج و لنكتب :

```
Console.WriteLine(xmlDoc.ToString);
```

حسناً لنعد إلى ثلاثة و ننفذ لنرى النتيجة : 1 – 2 – 3 و Ctrl+F5 ! : الخرج :

```
System.Xml.XmlDocument
```

للأسف فإن استخدام ToString هنا لن يجدي نفعاً إذا لنحاول مع تابع آخر خاص بهذا الأمر :

```
Console.WriteLine(xmlDoc.OuterXml);
```

الخرج سيكون على النحو التالي :

```
<BookParticipants><BookParticipant
type="Author"><FirstName>Joe</FirstName><LastName>Rattz</LastName></BookParti
cipant><BookParticipanttype="Editor"><FirstName>Ewan</FirstName><LastName>Buc
kingham</LastName></BookParticipant></BookParticipants>
```

أما لو عدنا إلى المثال التالي عندما كتبنا عناصر XML باستخدام الصف XElement و حاولنا استخدام ToString لإظهار النتائج سنحصل على النتيجة التالية :

```
<BookParticipants>

  <BookParticipant type="Author">
```

```
    <FirstName>Joe</FirstName>

    <LastName>Rattz</LastName>

  </BookParticipant>

  <BookParticipant type="Editor">

    <FirstName>Ewan</FirstName>

    <LastName>Buckingham</LastName>

  </BookParticipant>

</BookParticipants>
```

لاحظ الاختلاف في تنسيق الملف .

هذا بالنسبة للحفظ أما بالنسبة لتحميل ملفXML ما إلى برنامجنا سنقوم باستخدام التابع Load وهو تابع Static أي نستدعيه من أجل الصف و ليس من أجل كائن منه..بفرض أننا قمنا بإنشاء ملف في المشروع باسم TestFile و نريد تحميله الآن فإننا نكتب :

```
XDocument xmlDoc1 = XDocument.Load("TestFile.xml");
```

بالطبع فهنالك العديد من الخيارات أثناء تحميل الملف سنتكلم عنها لاحقاً.

بالعودة قليلاً إلى الصف XElement سنجد أننا استخدمنا الباني الذي يأخذ وسطين ، الأول يعبر عن اسم الـ Tag و الثاني يعبر عن المحتوى داخلها أي إذا أخذنا المثال التالي :

```
Console.WriteLine(new XElement("Name","Hammod"));
```

حيث سيكون الخرج :

```
<Name>Hammod</Name>
```

أما إذا أردنا القيمة المحتواة في هذا العنصر فلن نحتاج أكثر من استدعاء التابع Value :

```
XElement name = new XElement("Author", "Hammod");
Console.WriteLine(name.Value);
```

لننتقل الآن إلى ناحية أخرى فيLINQ و لنفترض أنه لدينا مصفوفة مكونة من مئة اسم نريد كتابتها على شكل ملف XML كما في المثال السابق تماماً . هذه عملية مضنية حقاً ! لذلك لابد من طريقة ما لكتابتها بسرعة و فاعلية و بأقل عدد ممكن من أسطر الكود، من المؤكد أنكم الآن تفكرون في الحلقات و لكن ما رأيكم بتجربة أمر آخر ؟ لنشاهد المثال التالي الذي يحوي أربعة أسماء و يمكن تعميمه لأعداد أكبر طبعاً :

```
string[] names = { "M.Hammod", "Golden Man", "Mohammed_807", "Nawwar(music
man)" };
XElement OurGroupMembers = new XElement("OurGroup",
```

```
                                                    from n in names
                                                    select new XElement("Name",n));
            Console.WriteLine(OurGroupMembers);
```

لاحظوا ما الذي قمنا به ، و كيف استطعنا بناء الملف باستعمال select من مصفوفة الأسماء و النتيجة التي ستظهر هي :

```
<OurGroup>

  <Name>M.Hammod</Name>

  <Name>Golden Man</Name>

  <Name>Mohammed_807</Name>

  <Name>Nawwar(music man)</Name>

</OurGroup>
```

لندخل في العمق قليلاً و لنفرض أننا قمنا بتغيير أحد الأسماء في المصفوفة بعد بناء الملف فما الذي سيحصل ؟

```
string[] names = { "M.Hammod", "Golden Man", "Mohammed_807", "Nawwar(music man)" };
XElement OurGroupMembers = new XElement("OurGroup",
                                                    from n in names
                                                    select new XElement("Name",n));
names[3] = "Mu_Nizar";
Console.WriteLine(OurGroupMembers);
```

الخرج سيكون معتمداً على الإجابة عن السؤال التالي : متى يتم بناء الملف ؟ هل يتم بناؤه مباشرة أم يتم تأجيله ؟ الإجابة أننا عندما نستخدم الصف XElement يتم ذلك مباشرة لذلك لن يظهر أي تغيير في بنية الملف :

```
<OurGroup>

  <Name>M.Hammod</Name>

  <Name>Golden Man</Name>

  <Name>Mohammed_807</Name>

  <Name>Nawwar(music man)</Name>

</OurGroup>
```

حسناً ما الحل إذا لتغيير بنية الملف باستعمال الكود السابق ؟ الحل باستخدام XStreamingElement بدلاً من XElement :

```
XStreamingElement OurGroupMembers1 = new XStreamingElement("OurGroup",
                                                    from n in names
```

```
                                            select new XElement("Name", n));
names[3] = "Mu_Nizar";
Console.WriteLine(OurGroupMembers1);
```

هنا سنحصل على التغيير المنشود و ستكون النتيجة :

```
<OurGroup>

  <Name>M.Hammod</Name>

  <Name>Golden Man</Name>

  <Name>Mohammed_807</Name>

  <Name>Mu_Nizar</Name>

</OurGroup>
```

كانت هذه مجرد مقدمة بسيطة عن LINQ to XML على الرغم من أننا لم نتحدث عن LINQ حتى الآن! و لكن أصبح لدينا فكرة جيدة عن إمكانية إنشاء مستندات XML بسهولة و يسر بواسطتها و الهدف أن نعلم أن LINQ ليست لمجرد الاستعلامات فقط. في المرة القادمة إن شاء الله سنتحدث بتفصيل أكبر عن LINQ و نكتب أمثلة لتبين لنا فعالية التعامل مع الـــ XML فيها.

Happy Programming!

HammooD 2010-03-19

m-khaled89@hotmail.com